DATABASES AND INFORMATION SYSTEMS
H.-M. Haav, A. Kalja and T. Robal (Eds.)
Proc. of the 11th International Baltic Conference, Baltic DB&IS 2014
TUT Press, 2014

Data Galaxies: A Common Space for Data Manipulation and Visualization

Sergejs KOZLOVICS [1,2] and Peteris RUCEVSKIS
Institute of Mathematics and Computer Science, University of Latvia

Abstract. Heterogeneous data in various formats are everywhere and can bring certain benefits if we could analyze them. Although certain tools exist to manipulate and analyze data, it is very difficult to ensure inter-operation between them to be able to use the whole range of their capabilities. Data galaxies is a new concept that is intended to provide a common space for different kinds of data manipulations and visualizations, where existing tools can be joined in together to participate in data manipulation flows for obtaining the desired result. The power of the approach is demonstrated by several use cases.

Keywords. data galaxies, data visualization, data manipulation, query, data transformation

Introduction

In the Information Age data management is no more a prerogative of IT professionals. End users now have access to various kinds of personal data such as data about bank transfers and credit card operations, purchases with loyalty cards, public transport smart card usage, mobile service usage, etc. Besides, there are public data like the data found in DBPedia and Europeana. Modern users are aware of those data and are able to query and analyze them to some extent. Companies storing user's data usually provide online tools for data querying and visualization, but these tools are tied to a particular domain (and even to a particular company) and their capabilities often are limited. For instance, it is unlikely that having an excerpt of the credit card operations for the last month, the banking system would be able to infer how much money was spent for sweets. Even when it was possible (e.g., by specifying that payments to "Laima" shops are for sweets), the user could still go further and ask: Is there a relationship between the day, time, and money spent to sweets? Probably, the user should avoid being near sweet shops during certain time of the day.

To show the need for advanced manipulations on data and limitations of existing approaches, we extend our scenario. Now, the user may be interested in data concerning sweets from all bank accounts of his family members. Obviously, online banking systems are not usable here because of the competition between banks, legal issues on data privacy, and for technical reasons. Thus, the user (the head of the family) needs to obtain

[1]Corresponding Author: Sergejs Kozlovics, Institute of Mathematics and Computer Science, Room 420, Raina blvd. 29, LV-1459, Riga, Latvia; E-mail: sergejs.kozlovics@lumii.lv.

[2]Supported by ESF project 2013/0005/1DP/1.1.1.2.0/13/APIA/VIAA/049

the data from all the family members (who may be using different banks providing the data in different formats) and use external tools to unify and merge those data. However, the tools suitable for data merge may not be suitable for queries or visualizations. Thus, the user needs another tool to query the data and, possibly, one more tool to visualize the result (e.g., as a pie chart). The scenario becomes even more complicated, if other family members are wishing to filter their data to hide their expenses except expenses for the sweets.

Next month, the user (the head of the family) may want to repeat the process to see how his family's expenses for sweets have changed. Here, a capability of storing user's actions to obtain the desired pie chart from raw data would come in handy.

The example above reveals that to obtain the desired result a series of manipulations on data have to be performed. They include[3]:

- obtaining data (from different sources and in different formats),
- integrating data (into a common ecosystem for further analysis),
- filtering data,
- data enrichment (i.e., improving data or extending them with additional semantic information);
- visualization,
- queries.

Our goal is to provide a common space for performing such manipulations on data and to store them for further execution on future data. We use the galaxy metaphor to represent such spaces. Each galaxy can represent several logically bound manipulations on data (like in the example above). In the next section we define data galaxies and their components (stars, planets, etc.). Then we mention several use cases and map them to the galaxy metaphor; this shows the necessity and sufficiency of all the galaxy components we have defined. Afterward we show how data galaxies can be implemented by means of the transformation-driven architecture, which is a system-building approach that uses models and model transformations at runtime. Finally, we discuss third-party tools that can be incorporated within data galaxies to perform manipulations on data using the capabilities of those tools.

1. Data Galaxies

First, we explain the technical space concept as a space for storing data and providing specific tools for manipulating data. Then, we define data galaxies and their components. Finally, we describe the life cycle of galaxies.

1.1. Technical Spaces

In 2002 I. Kurtev, J. Bézivin and M. Aksit have made an observation that there are multiple technologies that organize data into three levels of abstraction [2,3,4]. Data themselves lay at Level 1 (they are called a **model**). The description of data (or, the language, in which data are described) is called a **metamodel** (laying at Level 2) The language

[3]Discussions within the project "Scenario based semantic and graphical data processing technology". Similar manipulations with data are discussed under the data journalism topic [1].

Table 1. Some technical spaces and their meta-levels.

Technical space	Model	Metamodel	Meta-metamodel
Relational database	Database rows	ER-model (schema)	System tables
RDF/OWL	OWL individuals	OWL classes	OWL DL (or another OWL variant)
MOF-like	model (in XMI or other format [5])	metamodel (in XMI or other format)	MOF (or similar standard, e.g., ECore) [6,7]
XML	XML document [8]	XML schema (.xsd)	XML meta-schema (XSD.xsd)
Spreadsheet	rows of a spreadsheet document	sheet and column titles, cell types	document format (e.g., Excel or LibreOffice Calc)

for describing metamodels is a **meta-metamodel** (Level 3). A meta-metamodel is usually able to describe itself. Table 1 lists some technical spaces and the corresponding meta-levels.

We use the term **repository** to denote a store, where technical spaces store their data. Usually, model and metamodel are stored, while the meta-metamodel is fixed and considered to be known in advance. One technical space may have multiple repositories available. For instance, there are numerous database management systems from SQLite to Oracle. OWL ontologies can be stored in OWL files (in XML format), as RDF triples, or in some semantic repository such as StarDog, Sesame, Virtuoso, OWLIM, OWL API, or Apache Jena [4]. MOF-like models can be stored in the ECore format [7], or in repositories such as CDO[5], MetaMart Metadata Repository[6], and JR [9]. To find out more about technical spaces and different repositories within them, refer to our publication on model repositories [4].

Technical spaces are not just three meta-levels and different repositories. J. Bézivin and I. Kurtev explain [3]:

> "A technical space is a working context with a set of associated concepts, body of knowledge, tools, required skills, and possibilities. Apparently, there are human related components in this definition since most technologies have emerged in a given community that has knowledge, performs research, and even may have dedicated conferences. In addition, a technology allows creation and manipulation of artifacts."

Thus, one technical space may have a more convenient data format, while other technical spaces may have useful tools to perform manipulations on data, but the data need to be transformed to the appropriate format. Besides, while there may be several tools in different technical spaces providing the same functionality, the user may have skills of using a particular tool or knowledge in one particular technical space. This encourages us to use the best from different technical spaces by combining them. Data galaxies is our attempt to define a common space for integrating data from different technical spaces and for performing manipulations of those data.

[4]http://stardog.com/, http://www.openrdf.org/, http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/, http://www.ontotext.com/owlim, http://owlapi.sourceforge.net/, http://jena.apache.org/

[5]Connected Data Objects, http://www.eclipse.org/cdo/

[6]http://www.infolibcorp.com/metadata-management/software-tools/metadata-repository

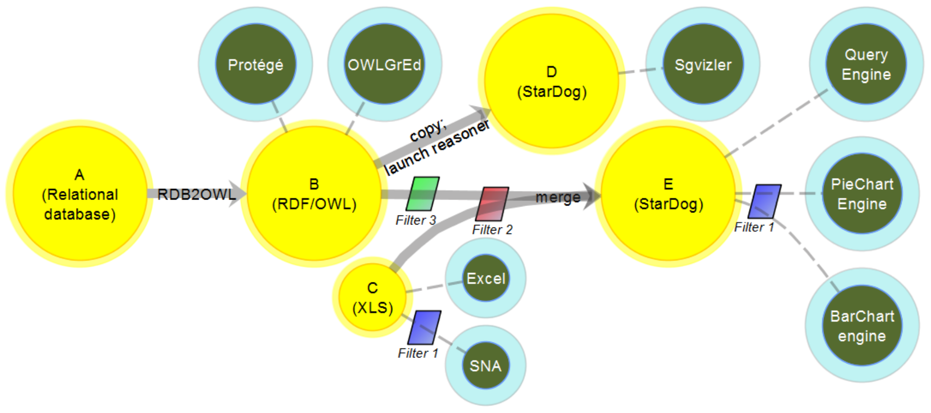

Figure 1. A data galaxy example. Bigger circles are *stars* associated with data repositories. Smaller circles are *planets* that provide visualizations and editors for the data from the stars. Planets have atmospheres, which are affected by data *filters* (parallelograms). Arrows are *stellar winds* that perform manipulations on data resulting in creation of new stars

1.2. The Data Galaxy Concept

By **data galaxy** we mean an interactive executable configuration of data storages (repositories) and manipulations with them. Figure 1 illustrates that. Among all the objects that exist in astronomical galaxies, for the purpose of our metaphor we concentrate on stars, planets, and stellar winds (star dust) within interstellar medium. Each **star** (a big circle) is associated with some repository from some technical space. The same repository may be referenced by multiple stars (this might be useful in certain cases).

Arrows describe the process of transferring data from one or more stars into another star, while performing some **manipulations** on them (e.g., copy a subset of data, merge two planets, replace certain data, etc.). This resembles how stars produce and disperse stellar winds, where dust particles reside, which can be used to produce new stars. Hence, we call the arrows **stellar winds**. For the sake of simplicity, each stellar wind has exactly one target star.. If still two target stars are required, two stellar winds can be involved. Stellar winds have direction that describes the flow of data, while performing manipulations. Since new manipulations can be performed after previous manipulations, stellar winds cannot form cycles. Thus, in a particular data galaxy, stars and stellar winds describe a directed acyclic graph (DAG).

Certain stars may have **planets** around them that allow the user to “touch” the data from the corresponding star. Planets are represented by certain plug-ins or external tools for visualizing, editing and querying data (we use the common word **visualization** to denote the functionality provided by planets). For instance, an OWL ontology can be edited either by the Protégé tool, or by means of the OWLGrEd tool (such tools are planets in our metaphor)[7]. Data containing graph-like structure can be visualized by means of the Social Network Analysis tool (SNA) or Tom Sawyer Perspectives[8]. Tabular data can be visualized by means of Microsoft Excel, LibreOffice Calc, Data Wranger, or OpenRefine.

[7]http://protege.stanford.edu/, http://owlgred.lumii.lv/
[8]http://sna.lumii.lv/, https://www.tomsawyer.com/products/perspectives/

We may think that the data from stars is caught by planets like energy from real stars is consumed by planets. To filter the radiation received from the stars, real planets use atmospheres. Similarly, in data galaxies planets also have atmospheres that are driven by data **filters**. Each filter can affect atmospheres of multiple planets (e.g., Filter 1 in Figure 1), thus, filters are actually cross-filters. A planet using a filter can display some GUI that allows the user to change the filter (e.g., to change the minimum and maximum boundaries for the data). This change can be used to alter the filter. Then the data galaxy synchronizes all the planets using that filter.

Filters can be also attached to stellar winds to guide them how new stars should be created. In this case, a filter is taken into a consideration when the corresponding stellar wind is executed (see Filters 2 and 3 in Figure 1).

In Astronomy, stars are classified into different types according to their luminosity effects and spectra. Similarly, each star in a data galaxy has its own type corresponding to the type of the repository that may be associated with that star (e.g., a relational database or an OWL ontology). Planets have also types corresponding to plug-ins or tools that are associated with them. The types of stellar winds correspond to types of data manipulations to be performed (e.g., merge). Each particular star, planet, or stellar wind can have a **configuration** (not shown in Figure 1) specific to the corresponding type. For instance, a star of the relational database type needs to know the database URL along with the user name and the password. An RDB2OWL transformation (stellar wind) requires a configuration file that describes how to map the database to the ontology. The SNA tool (a planet for visualizing graphs) needs to know how to map data columns to source and target vertices of the graph.

Since there can be numerous star types, planet types, and stellar wind types, the data galaxy application maintains the list of all these types. An out-of-box data galaxy application would contain certain predefined types, while other types could be developed by third-parties and attached to the galaxy application later. Some of these types can be adapters for calling external tools. The data galaxy application can be implemented in a way that allows new types to be registered at runtime. This would allow the user, for example, to configure a stellar wind and then use this configuration as type when creating a new stellar wind.

1.3. The Galaxy Life Cycle

Data galaxies are interactive. The user can connect to a new data source (repository) by creating a new star, or to re-connect an existing star to some other repository of the same type (the user may be asked for a star configuration). Then the user can explore/edit the data by introducing one or more planets. Afterward he can choose a source star and launch some stellar wind from it to obtain a new star with derived (refined) data. For instance, the user can choose to copy star data to a new star of different type (the "copy" stellar wind would ask for the target star type and its configuration). All the stars, planets, stellar winds, filters, and their configurations are stored in a galaxy for further re-execution on other data.

Data galaxies can be used in two modes. We have just described the designer's mode, where the user configures galaxies. The designer is an advanced user who is able to configure the whole galaxy. While prototyping, he can adjust all galaxy components (e.g., reconnect data to stars, and reconfigure stellar winds and filters) and also activate/deactivate planets and filters. The second mode is limited mode, where the galaxy is fixed and certain galaxy components are hidden. The user is able to configure just some of the components and view only some planets. For instance, the user may be allowed to specify the database name and credentials for a particular star to specify the source data. Also, the user can be allowed to modify some filters and view the resulting data through some final planets that have been left visible by the designer. All other components (planets, stars, stellar winds, and filters) are a "black box" in limited mode. The information on which components are hidden and which active/inactive is called a **run configuration**.

In both modes (designer's and limited), a special command (button) "Run" is available. This command traverses all the stars of the galaxy and executes the corresponding stellar winds in the topological order (since stars and stellar winds form a directed acyclic graph, this is possible). Thus, when the user reconfigures something (e.g., chooses another data source), he can launch the "Run" command to update the galaxy and to see the updated visualizations. The "Run" command takes a run configuration as an argument.

2. Use Cases

In this section we describe several use cases and map them to data galaxies.

Analyzing server log files to detect web-based attacks. Assume we have a web server that writes textual log files about web queries containing the port number, the source IP address, etc. Certain patterns in these log files may indicate an intrusion, e.g., when many different ports (say, > 1000) are being accessed (scanned) from the same IP address, or when too many queries from the same IP address are being issued in a short period of time. In this use case we want to detect such potential intrusions. For that, we create a data galaxy with two stars. The first star corresponds to a log-file. This star emits a stellar wind that corresponds to a Python program that generates a CSV file[9] and associates it with the second star. Then we can attach two planets to that star. The first planet would correspond to a notification engine that sends an e-mail (or an SMS) to the administrator when it detects suspicious activity (e.g., when the number of scanned ports exceeds the threshold of 1000). The second planet corresponds to the SNA tool, which is able to load a graph from CSV and visualize it. Thus, the server administrator can explore and filter the log data within SNA (e.g., to see only queries that cover at least 1000, or 64000 ports), and to make a decision how to resist the attack, if any. Now we can introduce two run configurations. The first configuration is intended to be used in a batch mode, i.e., to launch the galaxy automatically (to invoke its "Run" command) every 5 minutes[10]. In this run configuration, the notification planet is active, but the SNA planet is inactive (since SNA is an interactive tool). When the administrator receives a notification, he uses the second run configuration, where SNA is active. Thus, he can launch SNA to discover the problem and to make a decision.

Detection of unfair commercial practices. Assume that State News Agency has published information that Company C won certain government procurement. We want to check, whether there are certain cognate relations between individuals in the ruling party and Company C that could affect the results of the procurement. Assume that we

[9]CSV stands for Comma-Separated Values; a textual file, where data in each row are delimited by commas.
[10]This can be done by configuring some job scheduling service (e.g., *cron* in UNIX).

have access to certain databases, e.g., State Individuals Register (SIR), State Business Register (SBR), Political Party Register (PPR), State Procurement Register (SPR), and Party Financing Database (PFD). We can create a data galaxy with stars for the data from these databases (we can connect to a database directly, or export the required data to some data format and associate these exported data with a star, — this is determined by a star type). Then we can create a stellar wind that would transform the data from SIR to the RDF/OWL technical space using RDB2OWL or R2RML [10,11]. Then we can define certain rules (e.g., defined in Semantic Web Rule Language, SWRL [12]) to specify how to infer cognate relations (e.g., if A is a child of B and B is a child of C then A and C are relatives, etc.). A stellar wind taking these rules can infer which persons are relatives and put the inferred data into a new star (this is a kind of data enrichment). Using this information as well as information from SBR and PPR we can infer (by another stellar wind manipulation) which companies may be potentially related to certain parties. Then we can just check whether the money flows between SPR and PFD correlate with cognate relations found in the last step. Finally, we can summarize the results and put them into an Excel file by an appropriate stellar wind, which would create a new star (for the generated .xls) and attach a new Excel planet for viewing.

Going to opera. Assume the user wants to visit the opera theater. He goes to the opera web-site, where for each month he can find a table of all the productions to be performed in that month. The user is interested in productions to be held during either this month or the next two months. He has also certain restrictions: on the price (preferably less than 20 euro, but in certain cases he can consider the price up to 30 euro) and on the day of week (the user is always busy on Tuesday and Saturday evenings). Moreover, the user wants to visit a production that is accompanied by an orchestra and that receives the rating at least 4.5 from the online reviewers. For that, the user can create three stars corresponding to the three months. The stars would contain links to the corresponding opera web-pages with tables. For each of these stars, a HTML parsing stellar wind is launched that extracts the table and puts it into some database. Then all three databases are merged and for each production the currently available minimum and maximum ticket prices are appended from the ticket service web-page. Then the user can attach a planet (e.g., a visualizer for data tables) and some predefined filters: a range filter on the maximum price column (to specify the price below 20 euro), the value filter on the day of week column (to specify allowed values), a string value filter that checks whether the conductor column is not empty (to specify the productions accompanied by an orchestra), and a range filter to specify the minimum value of 4.5 for the rating column. The price filter can be adjusted, if the user decides to modify the price range.

In the first use case there are two users: the designer (who also has Python programming skills) and the administrator (who just executes the predefined galaxy in one of its two configurations). The other two use cases have just one user that acts as a designer.

It is also possible to define galaxies for the use cases mentioned in the introduction (we leave that as an exercise to the reader). Many other use cases can be represented by galaxies, too. Yet, the data galaxy concept stays simple: it is just a directed acyclic graph with certain planets and filters attached.

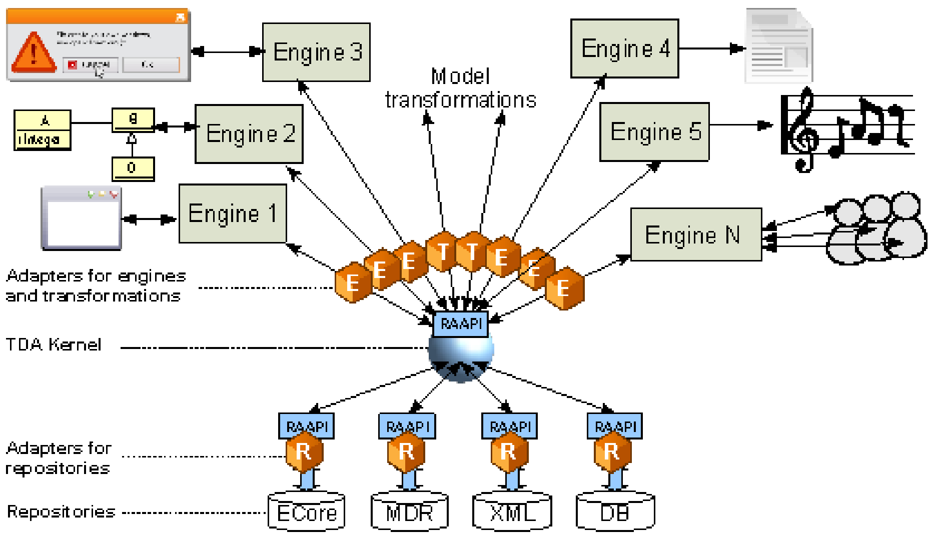

Figure 2. A technical view on the Transformation-Driven Architecture

3. Implementing Data Galaxies

Since 2008 we have been working on the Transformation-Driven Architecture, TDA, which is a software system building approach that uses models (from different technical spaces) and model transformations at runtime [13,14]. Although data galaxies can be implemented in various ways, TDA comes with many features that are essential for data galaxies.

3.1. TDA Features

In TDA, components implementing useful functionality for the application logic are called engines. Usually, engines provide graphical presentations and useful services [15,16,17,18]. Business logic is implemented by model transformations that “drive” the engines (Figure 2). TDA factors out many features that are useful for data galaxies:

- TDA has a built-in event/command mechanism, which ensures the communication between engines and transformations via special objects in the repository called events and commands;
- the TDA multi-repository mechanism can be used to mount multiple repositories from different technical spaces and to represent them as one big repository (for each repository type there must be an adapter for repository) [4]; this is useful to attaching data to stars;
- there is a universal repository access API (RAAPI) for different repositories; RAAPI can be used as an abstraction layer to perform certain manipulations on data regardless of the underlying technical space, e.g., copying all data from one repository to another;
- since interface metamodels of engines are programming-language neutral, engines and transformations can be written in different programming or transfor-

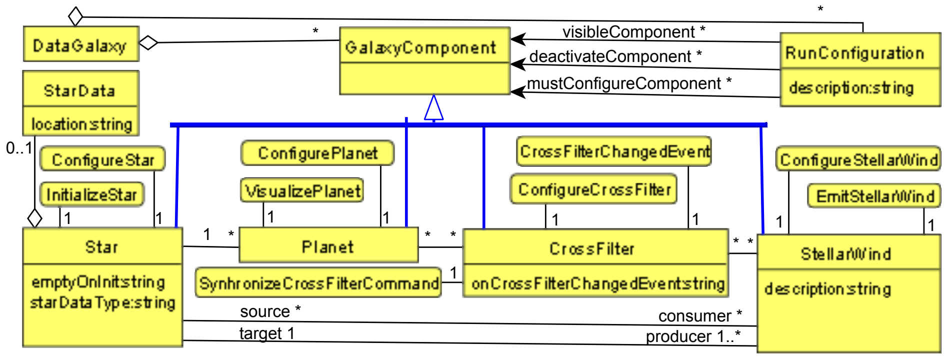

Figure 3. Galaxy Metamodel

mation languages (still, each such language needs an adapter); thus, planets and stellar winds can be written using different programming and/or transformation languages;

- TDA contains a universal built-in undo/redo mechanism that can be useful, when prototyping data galaxies.

3.2. *Implementing Galaxy Components*

Data galaxies themselves are implemented in Galaxy Engine that displays galaxies and allows users to design and run them. Galaxy Engine describes all the components according to Galaxy Metamodel (Figure 3). It contains all four types of galaxy components (*GalaxyComponent* subclasses). The *Star* component is attached to the *StarData* class representing data repositories. Since each star can be re-connected to a different repository, the *starDataType* attribute specifies one or more possible types of such repositories (types are associated with star data, since stars themselves are fixed, but their data are not; we talk on types below). The *emptyOnInit* attribute specifies whether the attached existing repository has to be emptied (non-existing repositories are always mounted as empty). When the user needs to create a star, Galaxy Engine issues a *ConfigureStar* command that asks for a star data type (if more than one) and calls the corresponding transformation that is used to configure the star (the type specific configuration can be defined in the type class, a *StarData* sublcass). The *InitializeStar* command calls a star data type specific transformation that initializes the star by mounting the required repository.

Planets also have two commands: a *ConfigurePlanet* command calls the planet type specific transformation that configures the corresponding planet, while the *VisualizePlanet* command calls the transformation that visualizes the given planet (transformations can attach TDA engines, which can represent certain tools or plug-ins). Stellar winds have two similar commands: *ConfigureStellarWind* is used to configure the given stellar wind, while *EmitStellarWind* launches the transformation implementing the stellar wind manipulation. The *CrossFilter* class can issue just one command *ConfigureCrossFilter* that can provide certain graphical dialog to configure the filter. When a filter is changed from the outside (e.g., from a planet using that filter), a command *SynchronizeCrossFilterCommand* must be issued. Then other planets and stellar winds using the same filter are notified about the change by means of the *CrossFilterChangedEvent*.

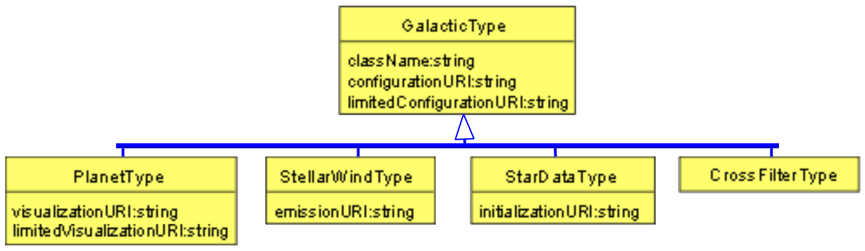

Figure 4. Metamodel of galactic types

The *RunConfiguration* class specifies run configurations. Among specifying a set of visible components (others will be invisible) and a set of inactive components (they are excluded from the galaxy when executing the "Run" command), a set of components that the user must re-configure prior launching the "Run" command can be specified (see the *mustConfigureComponent* association). For instance, the user may be required to configure certain stars specifying the initial data. Other visible components can be optionally re-configured by the user, too.

3.3. Types of Components Used Within Data Galaxies

Star data, planets, stellar winds and filters are of some type ("galactic type"). By convention, these types are subclasses of the corresponding classes in Figure 3. Since transformations that are called to execute galaxy commands (e.g., *ConfigurePlanet* or *EmitStellarWind*) differ from type to type, they are assigned to these galactic types.

Figure 4 shows the metamodel of galactic types. Each type has two URIs for specifying transformations for configuring the corresponding component[11]: *configurationURI* is used in the designer mode while *limitedConfigurationURI* is used in the limited mode since limited mode may involve certain restrictions (which might be configured in the designer's mode). *StarDataType* and *StellarWindType* have the additional transformation URIs for initializing a star and for emitting a stellar wind. The *PlanetType* class has two URIs for specifying visualization transformations: one is for the designer's mode while the other is for the limited mode (the visualization in the limited mode may show less, e.g., hide certain internal parameters).

4. Related Work

When J. Bézivin et al. proposed the technical space concept, they proposed also the projector concept — a transformation between two technical spaces. For instance, a MOF model can be transformed to XML and vice versa according to the XMI standard [5]. Such projectors can be used within data galaxies as stellar winds connecting star data from different technical spaces.

[11]In TDA, transformation name is a URI string consisting of two parts: the adapter name and the adapter-specific transformation (program) name, e.g., "python:MyManipulation.py". When a galaxy command is being issued, Galaxy Engine finds the assigned transformation URI and asks TDA to launch that transformation and to pass the command object as an argument.

There are numerous tools that can be used to manipulate and visualize data. OpenRefine, perhaps, is the most powerful tool that tries to cover almost the full data manipulation cycle[12]. It can process numerous data formats (and supports extensions), but focuses on tabular data representations. In the data galaxy world it can be a powerful planet for visualizing and refining tabular data. DataWrangler is another tool with multiple refine capabilities [19]. This tool is able to store the history of manipulations on data in the form of a script for further re-execution. In the data galaxy world, DataWrangler can be used to configure stellar winds corresponding to such scripts.

Crossfilter is a JavaScript library for filtering and visualizing data according to the given dimension. Freebase parallax is a browser interface for Freebase that is able to navigate from a set of data objects to a set of data objects, filter data, and plot charts. Sgvizler is a JavaScript library for the StarDog database being able to visualize results of SPARQL queries[13]. Facet Graphs is an approach for simplifying semantic queries for the end user [20]. All this software can be used as planets for the appropriate stars.

Data transformation languages such as RDB2OWL, R2RML, or XSLT can be used within stellar winds [10,11,21].

The import.io tool[14] is a tool for extracting data from web pages. The resulting data form a table. The tool has to be "trained" first. In the data galaxy world this tool can be considered a stellar wind that takes as input HTML data associated with some star. The "training" process can be launched during stellar wind configuration.

Popular spreadsheet processors like Microsoft Excel and LibreOffice Calc can be used as planets, as star data[15], or as a stellar wind executing VBA[16] script. Other third-party tools can also find their way to the data galaxy world.

Data galaxies can be viewed as a new kind of data ecosystems. Unlike existing ecosystems like Marinexplore and DataSift[17] that are tied to a particular domain (marine data, social data), our data galaxy approach allows the user to join data from different domains as well as to attach additional visualizations (not just predefined ones).

The Extract-Transform-Load (ETL) approach as well as application-centric hubs in artifact-centric business process modeling also resembles data galaxies [22,23]. However, these approaches focus just on data and their transformations, while data galaxies involve also visualization aspects implemented via planets and filters.

5. Conclusion

We presented data galaxies — a space, where data from different technical space can be mounted, manipulated, and visualized. This encourages using the best of available tools and technologies by joining them into a data galaxy. We believe that data galaxies provide the same environment for data from different technical spaces as Excel for tabular data. We are looking forward to implementing the data galaxy application as well as launching data galaxies in the web (in the style of Google Docs).

[12]http://openrefine.org/

[13]http://square.github.io/crossfilter/, https://code.google.com/p/freebase-parallax/, http://dev.data2000.no/sgvizler/

[14]https://import.io/

[15]The cell data can be accessed via Automation API in Excel or UNO API in LibreOffice.

[16]Visual Basic for Applications; LibreOffice Calc supports other scripting languages as well

[17]http://marinexplore.com/, http://datasift.com/

References

[1] Rogers S. Data journalism at the Guardian: what is it and how do we do it? [WWW] http://www.theguardian.com/news/datablog/2011/jul/28/data-journalism (accessed 06.05.2014).

[2] Kurtev I., Bézivin J., Aksit M. Technological spaces: An initial appraisal. In: *Proceedings of the 4th International Symposium on Distributed Objects and Applications, 30 October - 1 November 2002, Irvine, USA*. Springer-Verlag, 2002.

[3] Bézivin J., Kurtev I. Model-based technology integration with the technical space concept. In: *Metainformatics Symposium*, 2005.

[4] Kozlovics S. The orchestra of multiple model repositories. In: *SOFSEM 2013: Theory and Practice of Computer Science, 26-31 January 2013, Špindlerův Mlýn, Czech Republic.* Springer Berlin Heidelberg, 2013. 503–514. (LNCS; 7741).

[5] Object Management Group. OMG MOF 2 XMI Mapping Specification, Version 2.4.1 [WWW] http://www.omg.org/spec/XMI/2.4.1/ (accessed 06.05.2014).

[6] Object Management Group. OMG Meta Object Facility (MOF) Core Specification Version 2.4.1 [WWW] http://www.omg.org/spec/MOF/2.4.1/ (accessed 06.05.2014).

[7] Steinberg D., Budinsky F., Paternostro M., Merks E. *EMF: Eclipse Modeling Framework, 2nd Edition*. Addison-Wesley, 2008. 744 p.

[8] Extensible Markup Language (XML) 1.1 (Second Edition) [WWW] http://www.w3.org/TR/xml11/ (accessed 06.05.2014).

[9] Opmanis M., Čerāns K. Multilevel data repository for ontological and meta-modeling. In: *Selected Papers from the Ninth International Baltic Conference, DB&IS 2010.* Amsterdam: IOS Press, 2011. 125–138. (Frontiers in Artificial Intelligence and Applications; 224).

[10] Būmans G., Čerāns K. RDB2OWL: a practical approach for transforming RDB data into RDF/OWL. In: *Proceedings of the 6th International Conference on Semantic Systems (I-SEMANTICS '10), 1-3 September 2010, Graz, Austria.* New York, NY: ACM, 2010. 25:1–25:3.

[11] R2RML: RDB to RDF Mapping Language [WWW] http://www.w3.org/TR/r2rml/ (accessed 06.05.2014).

[12] SWRL: A Semantic Web Rule Language [WWW] http://www.w3.org/Submission/SWRL/ (accessed 06.05.2014).

[13] Barzdins J., Kozlovics S., Rencis E. The Transformation-Driven Architecture. In: *Proceedings of DSM'08 Workshop of OOPSLA 2008, 19-20 October 2008, Nashville, TN, USA.* University of Alabama at Birmingham, 2008. 60–63.

[14] Kozlovics S., Barzdins J. The Transformation-Driven Architecture for interactive systems. *Automatic Control and Computer Sciences*, 2013, 47(1/2013), 28–37.

[15] Kozlovics S. A Dialog Engine Metamodel for the Transformation-Driven Architecture. In: *Scientific Papers, University of Latvia*, 2010, 756, 151–170.

[16] Kozlovics S. Calculating The Layout For Dialog Windows Specified As Models. In: *Scientific Papers, University of Latvia*, 2012, 787, 106–124.

[17] Barzdins J., Cerans K., Kozlovics S., Rencis E., Zarins A. A Graph Diagram Engine for the Transformation-Driven Architecture. In: *Proceedings of MDDAUI 2009 Workshop of International Conference on Intelligent User Interfaces 2009, 8 February 2009, Sanibel Island, Florida, USA*. 29–32.

[18] Kozlovics S. A universal model-based solution for describing and handling errors. In: *Perspectives in Business Informatics Research.* Springer Berlin Heidelberg, 2011. 190–203. (LNBIP, 90).

[19] Kandel S., Paepcke A., Hellerstein J., Heer J. Wrangler: Interactive visual specification of data transformation scripts. In: *Proceedings of CHI'11.* New York, NY: ACM. 3363-3372.

[20] Heim P., Ertl T., Ziegler J. Facet graphs: Complex semantic querying made easy. In: *The Semantic Web: Research and Applications.* Springer Berlin Heidelberg, 2010. 288–302. (LNCS; 6088).

[21] XSL Transformations (XSLT), Version 3.0 (a working draft) [WWW] http://www.w3.org/TR/xslt-30/ (accessed 06.05.2014).

[22] Kimball R., Caserta J. *The Data Warehouse ETL Toolkit: Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data*. Wiley, 2004. 528 p.

[23] Hull R., Narendra N., Nigam A. Facilitating workflow interoperation using artifact-centric hubs. In: *Service-Oriented Computing.* Springer Berlin Heidelberg, 2009. 1–18. (LNCS, 5900).